

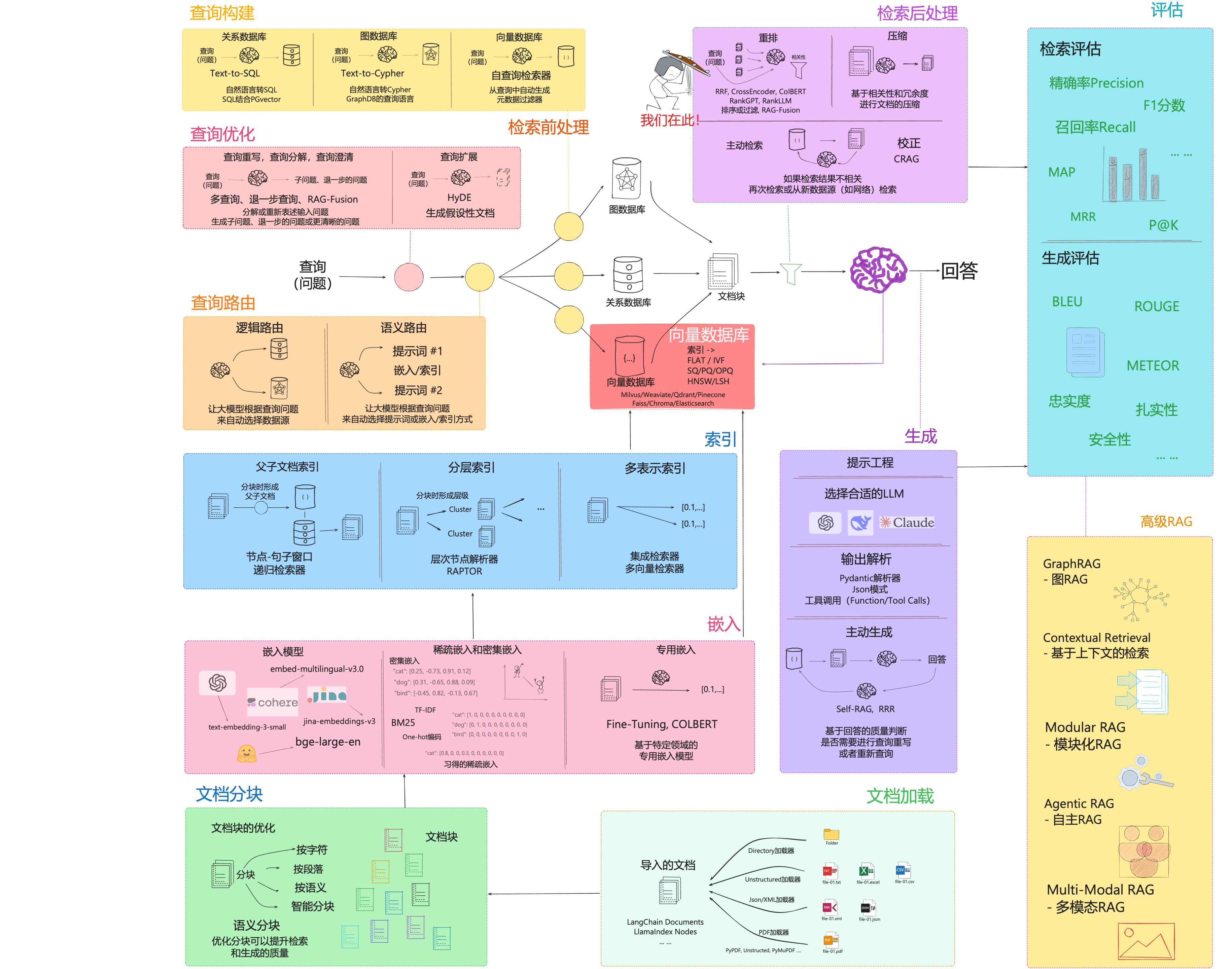
# 大模型RAG进阶实战营 – 组件篇

## 7 – 检索后处理

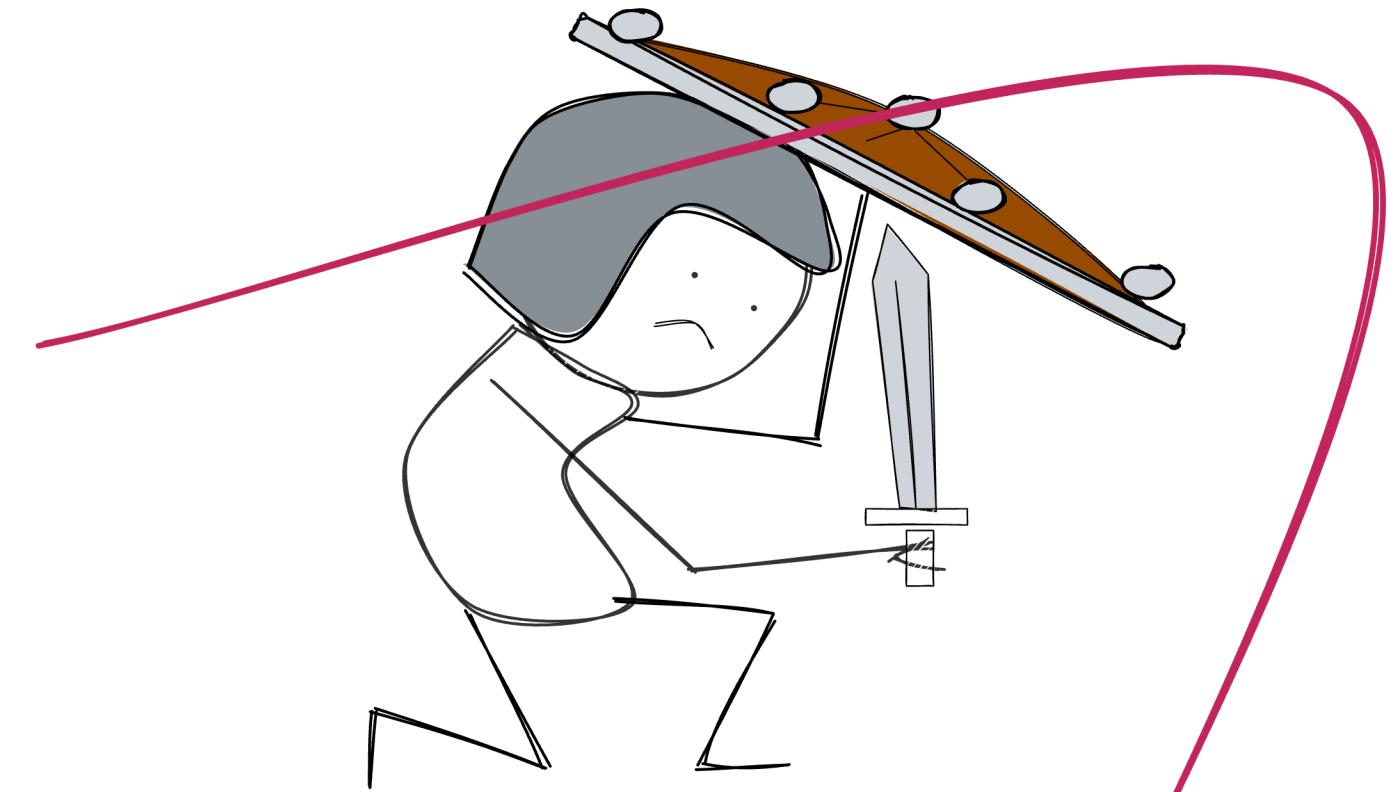
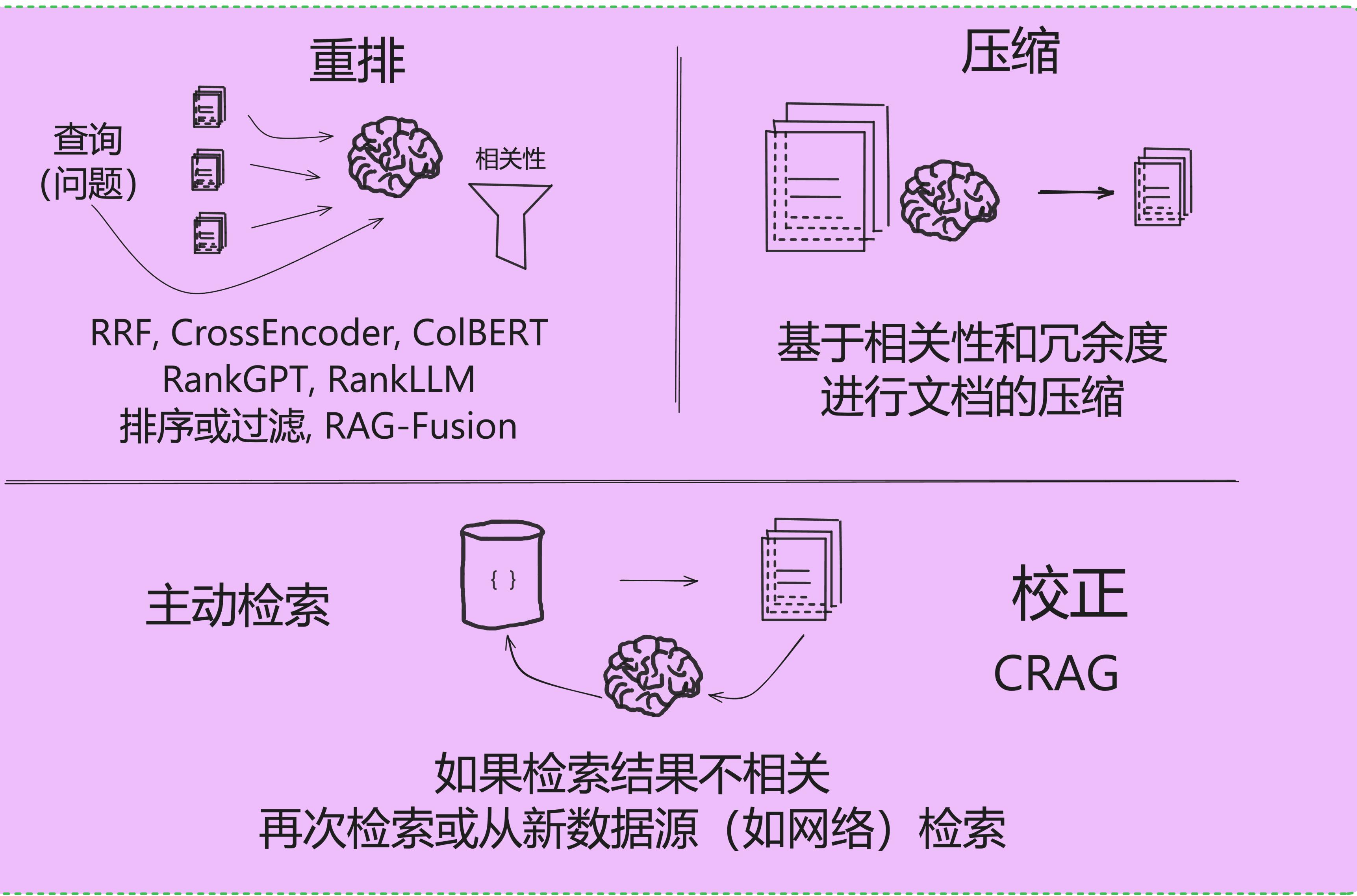
黄佳

新加坡科研局资深AI工程师

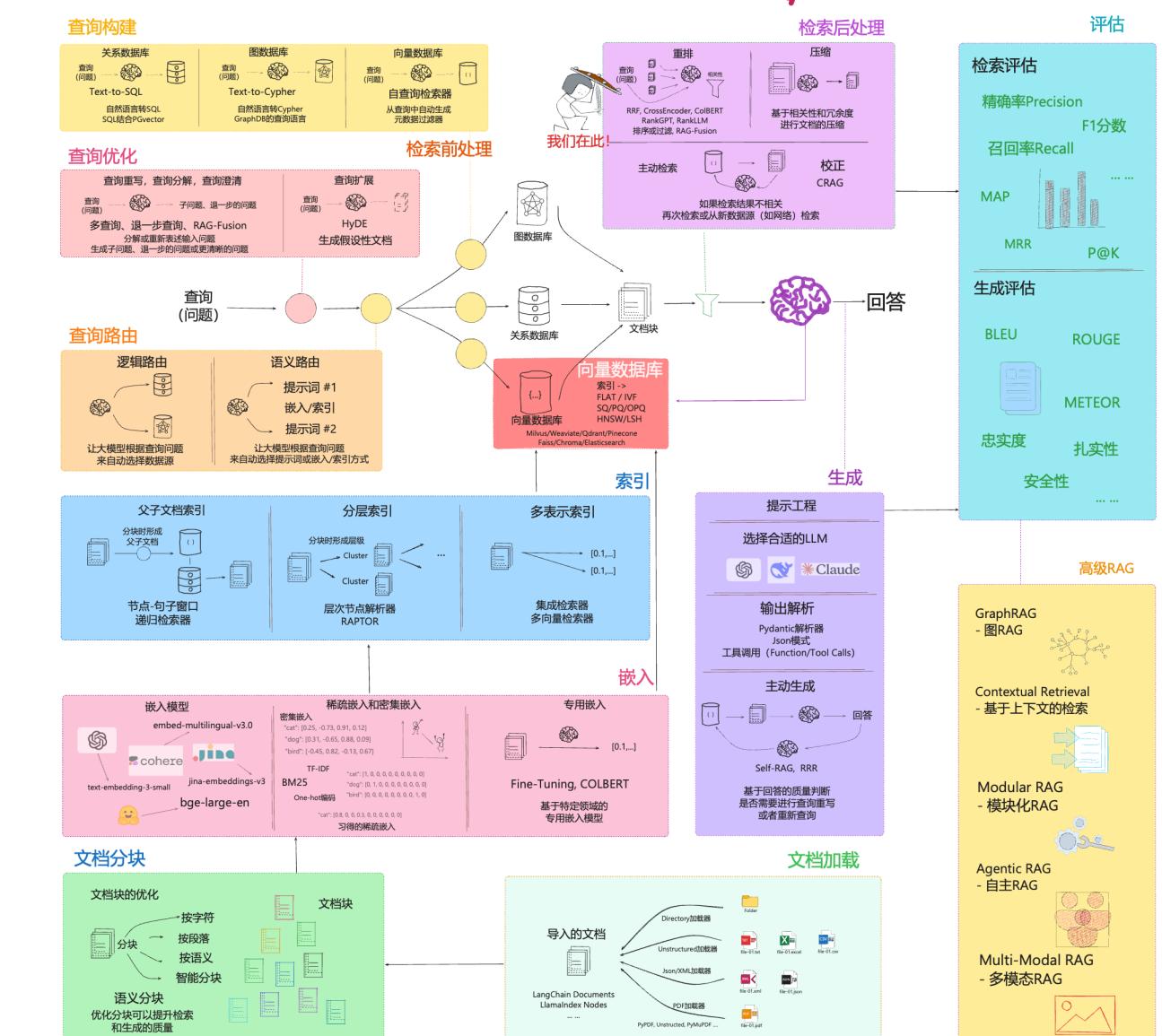




# 检索后处理



我们在此!



# 目录

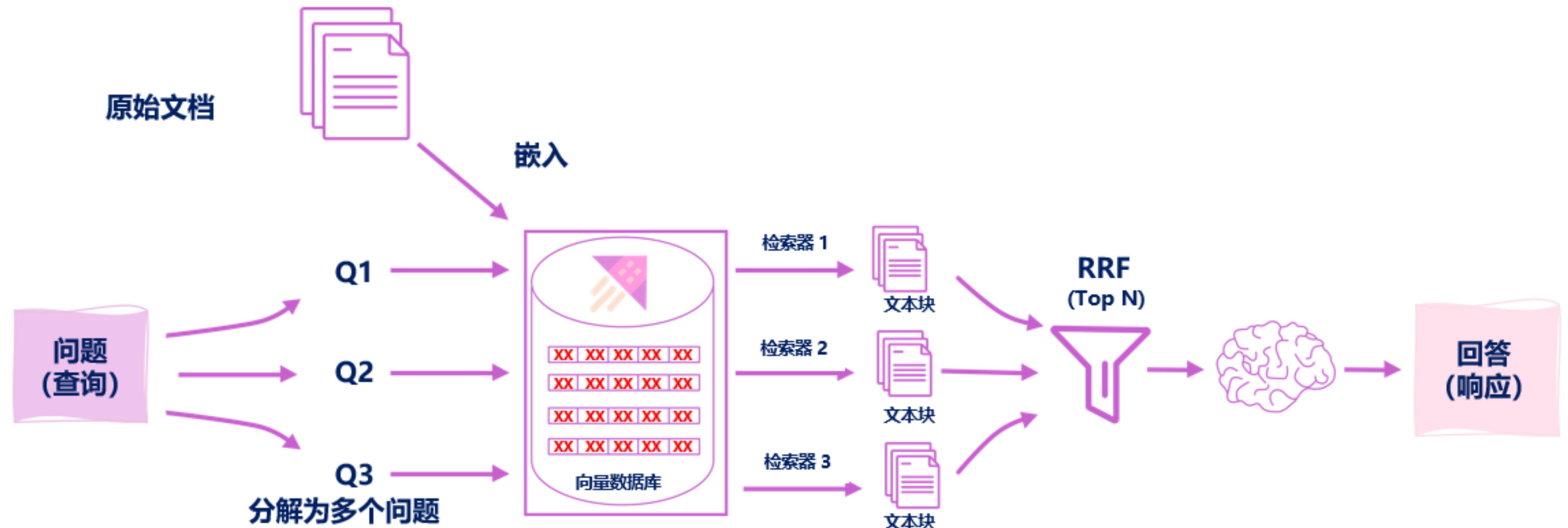
1. 重排技术——Rerank
2. 压缩技术——Compression
3. 校正技术——Correction

# 重排技术——Rerank

# 重排技术

- RRF重排
- Cross-Encoder重排
- CoBERT重排
- Cohere重排和Jina重排
- RankGPT和RankLLM
- 时效加权重排

# RRF-重排



# RRF：公式、代码和示例

$$\text{Score}_{RRF}(d) = \sum_{i=1}^N \frac{1}{\text{rank}_i(d) + k}$$

假设有两个排序器的输出

排序器1: "A", "B", "C"

排序器2: "C", "A", "B"

排序器1的贡献

$$\text{Score1("A")} = 1/(1 + 60) = 1/61$$

$$\text{Score1("B")} = 1/(2 + 60) = 1/62$$

$$\text{Score1("C")} = 1/(3 + 60) = 1/63$$

K = 60

排序器2的贡献

$$\text{Score2("C")} = 1/(1 + 60) = 1/61$$

$$\text{Score2("A")} = 1/(2 + 60) = 1/62$$

$$\text{Score2("B")} = 1/(3 + 60) = 1/63$$

融合得分

$$\text{ScoreRRF("A")} = 1/61 + 1/62$$

$$\text{ScoreRRF("B")} = 1/62 + 1/63$$

$$\text{ScoreRRF("C")} = 1/63 + 1/61$$

重新排序

"A" (ScoreRRF = 0.03252)

"C" (ScoreRRF = 0.03226)

"B" (ScoreRRF = 0.03200)

```
def reciprocal_rank_fusion(results: list[list], k=60):
    """
    RRF算法用于合并多个排序器的检索结果。
    参数:
    - results: 一个包含多个检索结果列表的列表，每个子列表代表一个排序器的输出
    每个子列表中的元素代表文档，按检索得分从高到低排序。
    - k: RRF公式中的参数，控制文档排名对融合分数的影响。默认值为60。
    返回:
    - reranked_results: 一个列表，包含经过RRF算法重新排序后的文档，按融合得分
    从高到低排序。
    """
    # 初始化一个字典，用于存储每个文档的融合得分
    fused_scores = {}
    # 遍历每个排序器的检索结果列表（即results中的每个子列表）
    for docs in results:
        # 遍历每个文档及其在该排序列表中的排名
        for rank, doc in enumerate(docs):
            # 将文档序列化为字符串格式，以便将其用作字典的键
            # 使用dumps(doc)将文档转换为字符串，便于在字典中存储
            doc_str = dumps(doc)
            # 如果该文档尚未在字典中出现，则初始化得分为0
            if doc_str not in fused_scores:
                fused_scores[doc_str] = 0
            # 根据RRF公式，计算当前文档的得分
            # rank是文档的排名（从0开始），k是常数参数，表示对排名的平滑处理
            fused_scores[doc_str] += 1 / (rank + k)
    # 将字典中的文档按照融合得分进行降序排序，得分高的文档排在前面
    reranked_results = [
        # 'loads(doc)' 将序列化的文档字符串还原为原始文档格式，配合对应的融合得
        # 分
        (loads(doc), score)
        # sorted函数对字典中的文档按得分进行排序，reverse=True表示按降序排列
        for doc, score in sorted(fused_scores.items(), key=lambda x: x[1], reverse=True)
    ]
    # 返回重新排序后的文档列表，其中每个元素是(文档, 融合得分)的元组
    return reranked_results
```

# Cross-Encoder重排

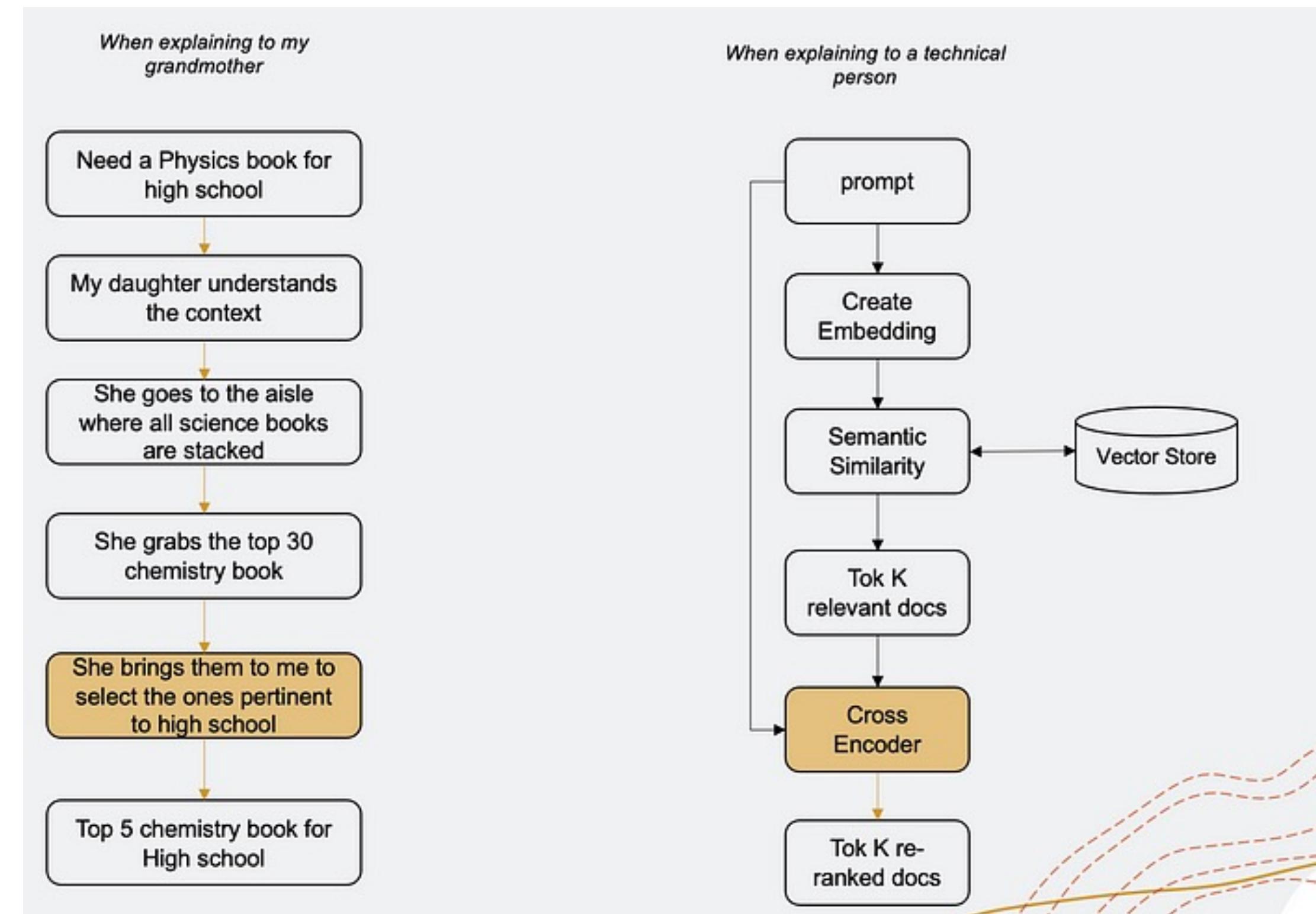


Cross-Encoder的输入拼接公式如下

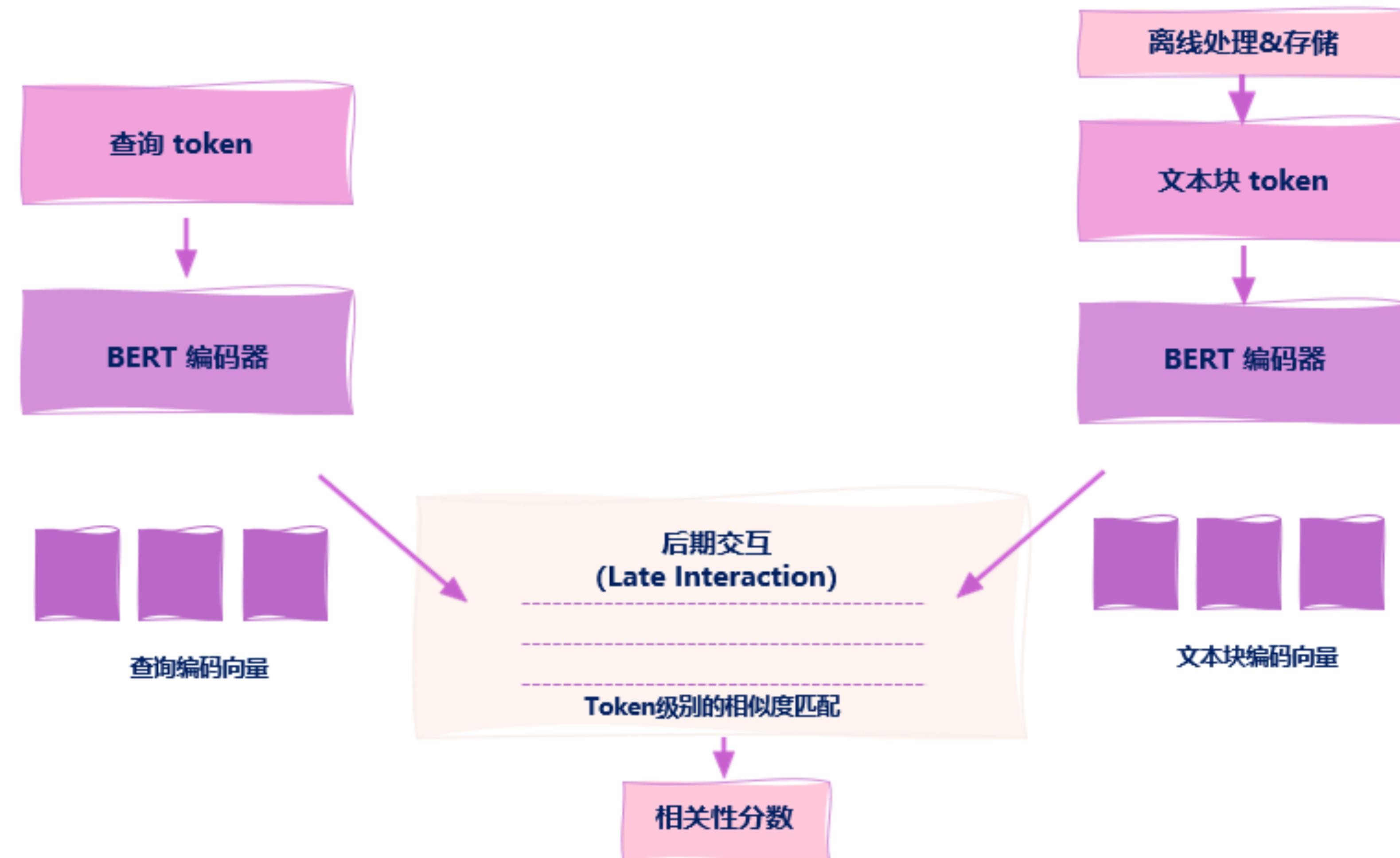
$$[CLS] + Query\ Tokens + [SEP] + Document\ Tokens$$

其中，[CLS]是分类标志，模型最终输出与其对应的向量，代表整体的相关性得分；[SEP]用于分隔查询和文档；Transformer的注意力机制允许查询和文档间的语义交互，捕捉复杂的上下文和深层次的语义匹配；经过简单的线性层或回归层处理后的[CLS]向量，会输出一个相关性分数。

# Cross-Encoder重排在RAG流程中的位置



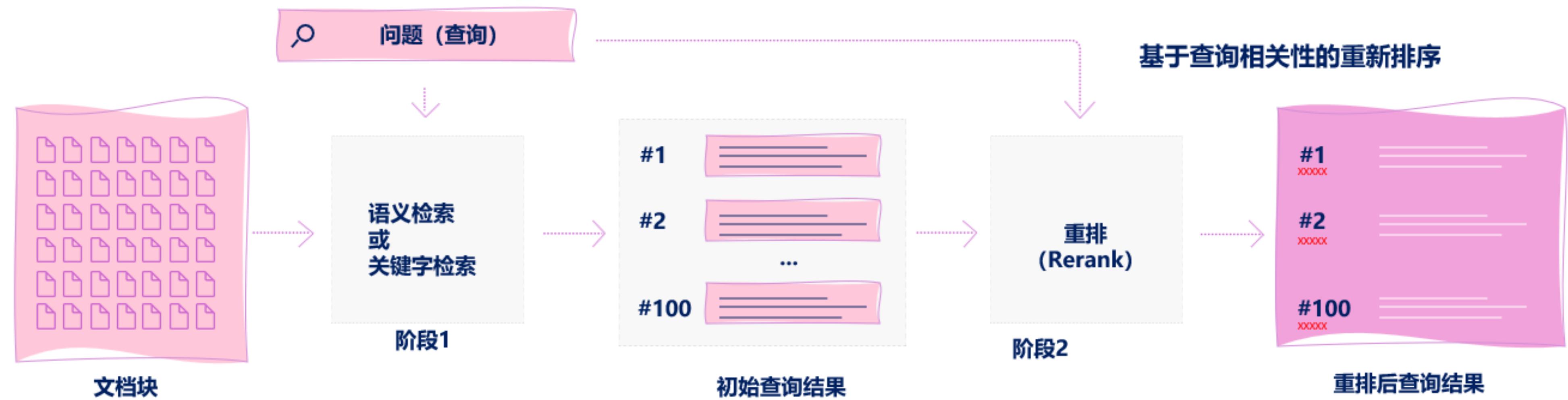
# Colbert重排



# CoBERT与Cross-Encoder、RRF的对比

特性	CoBERT	Cross-Encoder	RRF
设计目标	高效密集检索+精细重排	深度语义匹配，用于精细重排	融合多模型结果的轻量级重排
语义交互	token级别交互，捕捉细粒度语义	查询-文档全句交互，精确语义匹配	无语义交互，基于排名融合
计算成本	中等（查询与文档分离编码+点积）	高（每个查询-文档都需要进行全模型推理）	低（直接融合已有排名分数）
适用场景	密集检索或Top-k文档重排	Top-k文档精排	信号融合或轻量重排
上下文感知能力	强（基于token级别嵌入）	非常强（全句语义建模）	弱
适合大规模检索	是（支持预计算文档向量）	否（计算成本过高）	是

# Cohere重排和Jina重排



# RankGPT和RankLLM

## Is ChatGPT Good at Search? Investigating Large Language Models as Re-Ranking Agents

Weiwei Sun<sup>1\*</sup> Lingyong Yan<sup>2</sup> Xinyu Ma<sup>2</sup> Shuaiqiang Wang<sup>2</sup>  
Pengjie Ren<sup>1</sup> Zhumin Chen<sup>1</sup> Dawei Yin<sup>2†</sup> Zhaochun Ren<sup>3†</sup>

<sup>1</sup>Shandong University, Qingdao, China <sup>2</sup>Baidu Inc., Beijing, China

<sup>3</sup>Leiden University, Leiden, The Netherlands

{sunnweiwei,lingyongy,xinyuma2016,shqiang.wang}@gmail.com  
{renpengjie,chenzhumin}@sdu.edu.cn yindawei@acm.org  
z.ren@liacs.leidenuniv.nl

### Abstract

Large Language Models (LLMs) have demonstrated remarkable zero-shot generalization across various language-related tasks, including search engines. However, existing work utilizes the generative ability of LLMs for Information Retrieval (IR) rather than direct passage ranking. The discrepancy between the pre-training objectives of LLMs and the ranking objective poses another challenge. In this paper, we first investigate generative LLMs such as ChatGPT and GPT-4 for relevance ranking in IR. Surprisingly, our experiments reveal that properly instructed LLMs can deliver competitive, even superior results to state-of-the-art supervised methods on popular IR benchmarks.

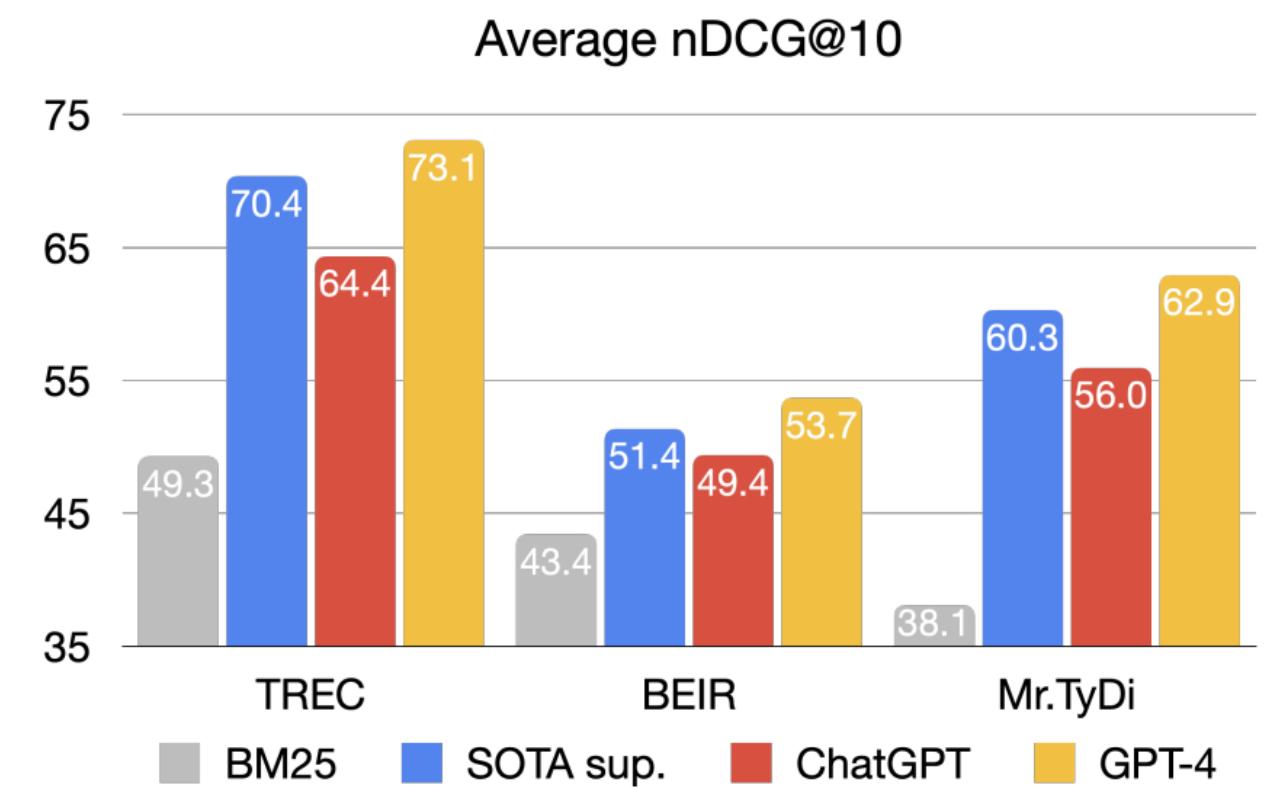


Figure 1: Average results of ChatGPT and GPT-4 (zero-shot) on passage re-ranking benchmarks (TREC, BEIR, and Mr.TyDi), compared with BM25 and previous best-supervised systems (SOTA sup., e.g., monoT5 (Nogueira et al., 2020)).

RankGPT和RankLLM都展示了将大模型应用于重排任务的潜力。前者强调零样本能力，即直接利用预训练模型就能有效完成重排任务，而不需要任何额外的训练步骤。相比之下，后者则更加关注通过微调开源模型来进一步提升在特定任务上的性能，从而满足更加专业化的需求。

# 时效加权重排：Time-weighted Vector Store Retriever

$$\text{score} = \text{semantic\_similarity} + (1.0 - \text{decay\_rate})^{\text{hours\_passed}}$$

## 时间加权检索器

时间加权检索器是一种除了相似性之外还考虑新近度的检索器。其评分算法如下：

```
let score = (1.0 - this.decayRate) ** hoursPassed + vectorRelevance;
```

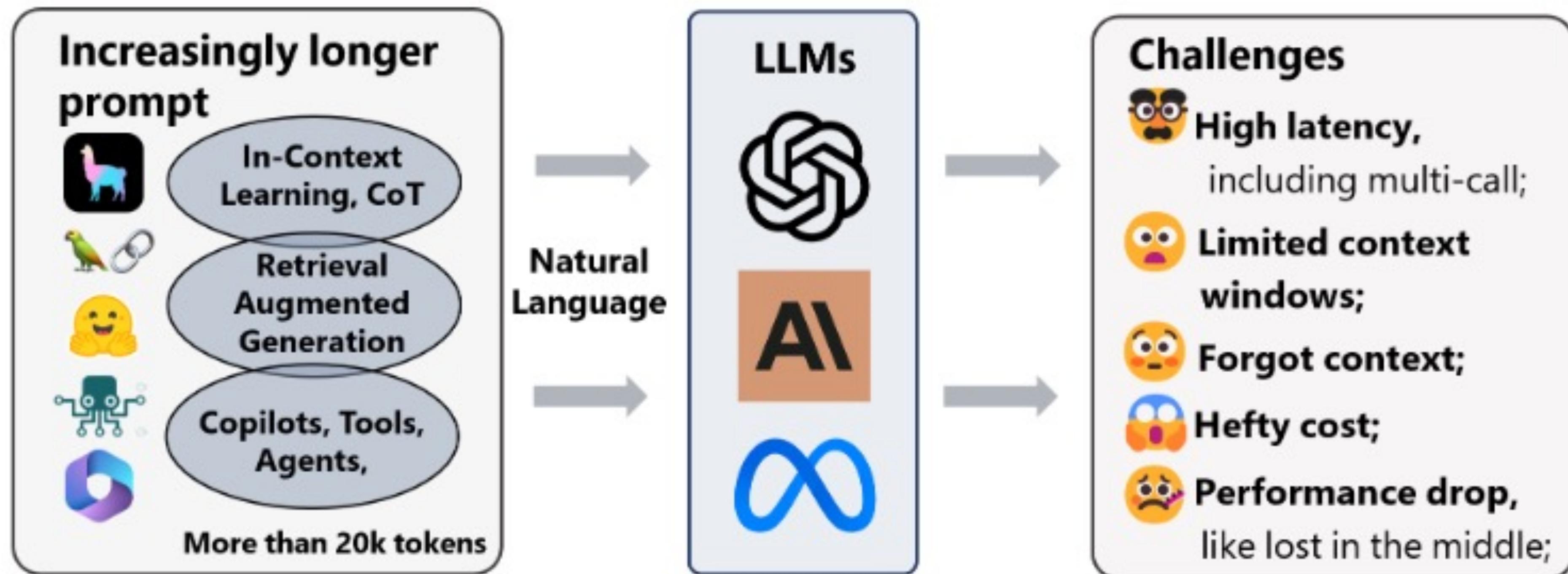
值得注意的是，`hoursPassed`以上时间指的是检索器中对象上次访问的时间，而不是创建的时间。这意味着经常访问的对象保持“新鲜”，得分更高。

`this.decayRate`是 0 到 1 之间的可配置十进制数。数字越小，表示文档将被“记住”更长时间，而数字越大，则表示最近访问的文档具有更大的权重。

请注意，将衰减率设置为恰好 0 或 1 会使得`hoursPassed`无关紧要，并使此检索器等同于标准向量查找。

# 压缩技术——Compression

# 压缩的必要性



# 压缩技术

LangChain的Contextual Compression Retriever上下文压缩检索器

LlamaIndex的Sentence Embedding Optimizer句子嵌入优化器

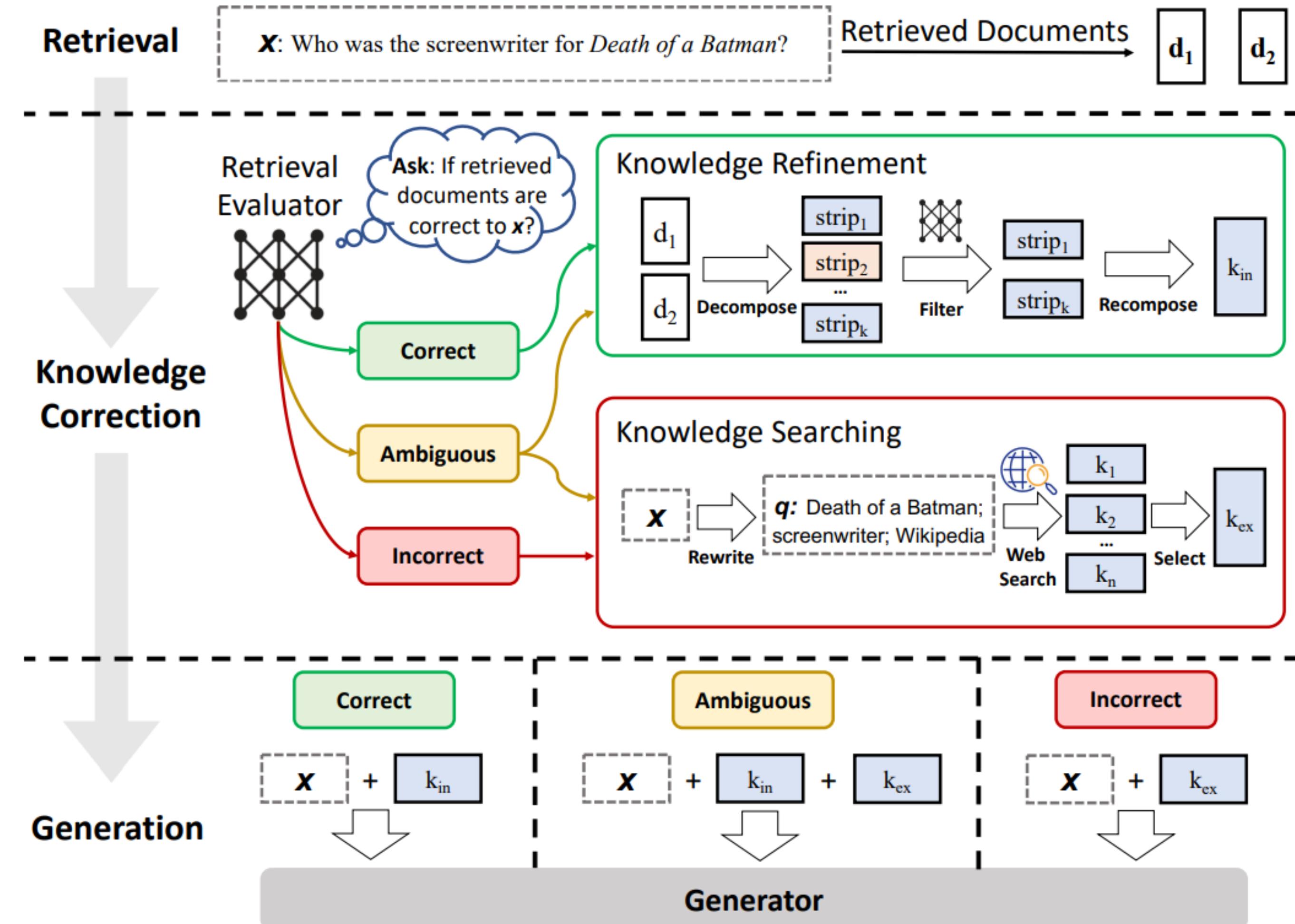
利用LLMLingua压缩提示词

RECOMP压缩方法

通过Prompt Caching记忆长上下文

校正技术——Correction

# C-RAG - Corrective RAG



# THANKS

